Exercice 1 : Couverture d'ensemble

CCINP type A

On considère le problème d'optimisation SETCOVERO suivant :

$$\begin{cases} \textbf{Entrée} : \text{Un ensemble } E \text{ fini et } E_1, ..., E_n \text{ des sous ensembles de } E. \\ \textbf{Solution} : \text{Une couverture de } E, \text{ i.e. un sous ensemble } I \subset [\![1, n]\!] \text{ tel que } \bigcup_{i \in I} E_i = E. \\ \textbf{Optimisation} : \text{Minimiser } |I| \text{ (qu'on appelle } \textit{taille de la couverture}). \end{cases}$$

Q. 1 Décrire le problème de décision SETCOVER associé à SETCOVERO.

Q. 2 Montrer que le problème SETCOVER est dans NP.

On dit qu'un sous-ensemble X de sommets d'un graphe G est une couverture par sommets de G si toute arête de G a au moins l'une de ses extrémités dans X. On admet dans la suite la NP-complétude du problème de décision VERTEXCOVER décrit comme suit :

$$\begin{cases} \textbf{Entrée} : \text{Un graphe } G \text{ et un entier } k. \\ \textbf{Question} : \text{Existe-t-il une couverture par sommets de } G \text{ de taille inférieure à } k \text{ ?} \end{cases}$$

Pour toute instance $I_V = (G = (S, A), k)$ de VERTEXCOVER, on associe l'instance I_S de SETCOVER définie par : $E = A$ et les sous ensembles de E sont les $E_s = \{a \in A \mid a \text{ est incidente à } s\}$ pour tout $s \in S$.

Q. 3 Montrer que si I_V est une instance positive de VERTEXCOVER alors I_S est une instance positive de SETCOVER. La réciproque est-elle vraie ? Justifier.

Q. 4 Montrer que le problème SETCOVER est NP-complet.

On se contente donc de résoudre le problème SETCOVERO de manière approchée au moyen de l'algorithme ci-dessous.

```
1 I ← ∅ et U ← E ;
2 while U ≠ ∅ do
3     if U n'intersecte aucun E_i then
4         retourner "couverture impossible" ;
5     else
6         Déterminer i tel que E_i ∩ U est maximal ;
7         Ajouter i à I ;
8         U ← U \ E_i ;
9 retourner I
```

Q. 5 On considère l'instance suivante de SETCOVERO : $E = [\![1, 8]\!]$, $E_1 = \{5, 6, 7, 8\}$, $E_2 = \{3, 4\}$, $E_3 = \{1\}$, $E_4 = \{2\}$, $E_5 = \{1, 3, 5, 7\}$, $E_6 = \{2, 4, 6, 8\}$. Montrer que cet algorithme glouton peut renvoyer une couverture de taille 4. Quelle est la taille de la couverture optimale ?

Q. 6 Montrer que cet algorithme est polynomial. On explicitera auparavant des choix de structures pour représenter I, U, E et les E_i.

Q. 7 En s'inspirant de la question **Q. 5**, montrer que cet algorithme n'est pas un algorithme d'approximation à facteur constant pour SETCOVERO.

Exercice 2 : File cyclique en OCaml

CCINP type B, 2023

Consignes : Ce sujet est à traiter en OCaml en complétant le fichier compagnon qui vous est fourni, nommé `ccinp_2023_file_cyclique.ml`. Outre des définitions de types et de valeurs, ce fichier contient une fonction d'affichage.

Le but de cet exercice est d'implémenter en OCaml une structure de file cyclique. Pour se représenter le fonctionnement d'une telle file on peut imaginer une pioche de cartes face cachée : celle sur le dessus du paquet est la tête. L'opération de défilement consiste à retourner cette carte, ainsi on découvre sa valeur, puis à la remettre sous le paquet. Si la pioche contient 5 cartes, en répétant cette opération 5 fois on est revenu à la situation de départ. Idem pour une file contenant 5 éléments : en effectuant 5 défilements la file est revenue à son état initial.

Les opérations. Les opérations de la file cyclique sont les mêmes que celles de la file, à ceci près que l'opération de défilement ne supprime pas l'élément en tête de la structure : elle le laisse en place pour le retrouver "au prochain tour". On dispose d'une opération spécifique pour supprimer l'élément en tête. L'opération d'ajout consiste à ajouter un élément après la queue (et donc juste avant la tête).

Les figures ci-dessous illustrent ces trois opérations, l'élément pointé étant la queue de la file.

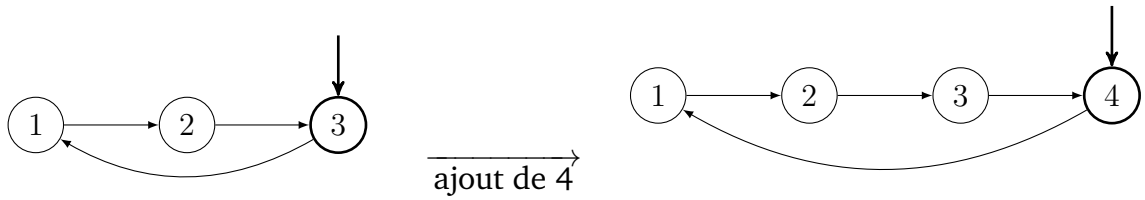

Figure 1 : Transformation d'une file cyclique par ajout

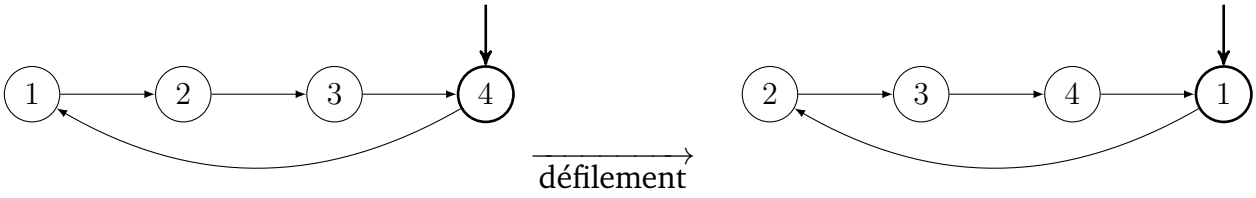

Figure 2 : Transformation d'une file cyclique par défilement

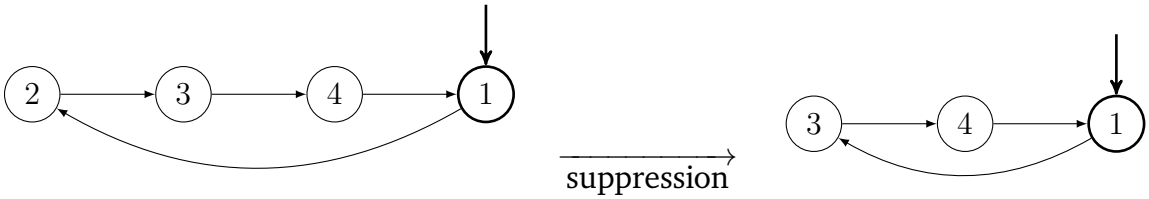

Figure 3 : Transformation d'une file cyclique par suppression de sa tête

Implémentation par maillon. On implémente ici la structure de file cyclique à l'aide de maillons. Chaque maillon correspond à un élément de la file, il contient non seulement cet élément mais aussi le maillon qui le suit dans la file. Ce maillon suivant est mutable pour permettre l'ajout et la suppression. On introduit donc le type suivant, où `'a` désigne le type des éléments.

```
type 'a maillon = {
  elem : 'a ;
  mutable suiv : 'a maillon
}
```

On peut alors décrire une file cyclique en indiquant le maillon correspondant à l'élément en queue, à moins que la file ne soit vide. On utilise un type option pour gérer cette disjonction de cas. De plus, afin de pouvoir modifier une file, on utilise une référence.

```
type 'a file = (('a maillon) option) ref
```

Q. 1 À la manière des figures ci-dessus dessiner les files `f1` et `f2` définies dans le fichier compagnon.

Q. 2 Compléter la fonction `cree_file_vide : unit -> 'a file` qui crée une file vide.

Q. 3 Compléter la fonction `est_file_vide : 'a file -> bool` qui teste si une file est vide.

Q. 4 Compléter la fonction `tete : 'a file -> 'a` qui calcule l'élément en tête d'une file non vide.

Q. 5 Compléter la fonction `defile : 'a file -> 'a` qui réalise le défilement pour une file cyclique.

Q. 6 Compléter la fonction `cree_singleton : 'a -> 'a file` qui crée une file réduite à un maillon contenant la valeur passée en argument et qui point vers lui même.

Q. 7 Compléter la fonction `ajoute : 'a file -> 'a -> unit` qui ajoute un élément dans une file. Préciser sa complexité.

Q. 8 Expliquer ce qui se passe lors du test d'égalité `f3 = (cree_singleton 1)`.

Q. 9 Compléter la fonction `est_singleton : 'a -> 'a file` qui teste si une file contient exactement un élément à l'aide du test d'égalité physique que permet l'opérateur `==`.

Q. 10 Compléter la fonction `supprime : 'a file -> 'a` qui supprime l'élément en tête d'une file non vide. On peut commencer par traiter le cas d'une file qui n'est pas un singleton.